AFRL-IF-RS-TR-2003-245
Final Technical Report
October 2003

REACTIVE SENSOR NETWORKS (RSN)

Penn State University

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. H584

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2003-245 has been reviewed and is approved for publication.

APPROVED: /s/
BRADLEY J. HARNISH
Project Engineer

FOR THE DIRECTOR: /s/
WARREN H. DEBANY, JR., Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	OCTOBER 2003	Final Jul 99 – May 03

4. TITLE AND SUBTITLE
REACTIVE SENSOR NETWORKS (RSN)

5. FUNDING NUMBERS
C - F30602-99-2-0520
PE - 62301E
PR - H584
TA - 16
WU - 01

6. AUTHOR(S)
Richard Brooks

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Penn State University
PO Box 30
State College Pennsylvania 16804

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFGA
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2003-245

11. SUPPLEMENTARY NOTES
AFRL Project Engineer: Bradley J. Harnish/IFGA/(315) 330-1884/ Bradley.Harnish@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***
The Reactive Sensor Networks project was based on a vision of sensor networks as virtual enterprises. A sensor network would be fielded by the military with certain goals in mind. The chaotic nature of the battlespace guarantees that mission parameters will change. The network would have to adapt in response to environmental constraints. RSN developed multiple technologies to enable the network to adapt to changing mission parameters: a) mobile code daemons support remote tasking of nodes and autonomous software reconfiguration. b) a distributed target-tracking algorithm enables the system to accurately track multiple targets with no single points of failure. c) autonomous configuration of target classification software uses the mobile code infrastructure to choose the classification algorithms that best suit the current mix of targets in the field. d) multiple sensing modalities were integrated into a tracking system at the PSU/ARL sensor network testbed. e) a distributed target counting application was derived and fielded. All approaches were implemented, tested and validated. The project also produced a new design approach for networks of embedded systems using cellular automata based tools. From our experience, these tools support an exploratory approach to system design that is very appropriate for use in implementing sensor networks.

14. SUBJECT TERMS
Sensor Networks, Mobile Code, Distributed Target Tracking and Classification

15. NUMBER OF PAGES
41

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

Table of Contents

List of Figures

List of Tables

EXECUTIVE SUMMARY

The Reactive Sensor Networks (RSN) project was funded as part of the DARPA IXO Sensor Information Technology (SensIT) program from July 1999 to May 2003. It consisted of a base project and one year extension. The program was based on a vision of sensor networks as virtual enterprises. A sensor network would be fielded by the military with certain goals in mind. The chaotic nature of the battlespace guarantees that mission parameters will change. The network would have to adapt in response to environmental constraints.

RSN developed multiple technologies to enable the network to adapt to changing mission parameters:

- Mobile code daemons support remote tasking of nodes and autonomous software reconfiguration.
- A distributed target-tracking algorithm enables the system to accurately track multiple targets with no single points of failure.
- Autonomous configuration of target classification software uses the mobile code infrastructure to choose the classification algorithms that best suit the current mix of targets in the field.
- Multiple sensing modalities were integrated into a tracking system at the PSU/ARL sensor network testbed.
- A distributed target counting application was derived and fielded. Analysis found that the density of sensors in the network limit the ability of the network to distinguish between targets.

All approaches were implemented, tested and validated. The PI and ARL engineers participated in an exercise at 29 Palms Marine Training Grounds. Demonstrations were given at multiple PI meetings. This included capstone projects. As the result of a Defense University Research Instrumentation Program (DURIP) project the PI was able to implement a testbed at PSU/ARL, where the technologies were thoroughly analyzed.

In addition, RSN produced a new design approach for networks of embedded systems. It is difficult to design scalable algorithms for these systems, since the algorithms need to accept random failures and environmental disruptions. To counter this, we have explored the use of cellular automata based tools. From our experience, these tools support an exploratory approach to system design that is very appropriate for use in implementing sensor networks.

1. BACKGROUND

Embedded sensors are now part of large loosely coupled networks with mobile code and data. Data supply and demand are stochastic and unpredictable. Processing occurs concurrently on multiple processors. Applications are in hostile environments with noise-corrupted communication and failure prone components. Under these conditions, efficient operation cannot rely on static plans. Van Creveld has defined five characteristics hierarchical systems need to adapt to this type of environment [van Crefeld 1985,Czerwinski 1998]:

- Decision thresholds fixed far down in the hierarchy.
- Self-contained units exist at a low level.
- Information circulates from the bottom up and the top down.
- Commanders actively seek data to supplement routine reports.
- Informal communications are necessary.

Organizations based on this approach have been successful in market economies, war, and law enforcement [Cebrowski 1998]. Our approach is influenced by and consistent with these points.

Data requests *pull* data across the network. Sensors *push* information by providing timely data. Requests and data follow paths of least resistance through intermediate nodes. Mobile data and code allow signal processing, Automatic Target Recognition (ATR), and target tracking to be done during data transmission. Processing is done robustly and efficiently by giving each node the ability to make limited local decisions using locally available information.

The Reactive Sensor Networks (RSN) project implement a prototype of this approach, combining the following technologies:

- Mobile code and data support
- Integration of multiple sensing modalities
- Autonomous software configuration
- Distributed target tracking
- Distributed target counting

The result was a distributed virtual enterprise that developed data flows in response to observed behaviors in the environment. Prototype implementations of these technologies were implemented and tested. Tests were performed both at 29 Palms Marine Training Ground and at testbeds designated by DARPA IXO management.

Results from analytical work was combined with experimental testing to reach the following conclusions:

- Self-organizing systems like the one proposed will be more dependable than systems with static hierarchies.
- Data association for distributed target tracking implementations has a lower computational complexity than centralized approaches.

- The amount of data traffic required for large scale distributed tracking implementations should be less than that required for centralized tracking approaches under situations like the test implementation at 29 Palms.
- The ability of these systems to infer information from their environment is limited by data sampling rates, which are in turn limited by the density of sensor platforms in the battlespace.

2. STATEMENT OF PROGRAM OBJECTIVES

RSN's objective was to create and implement sensor network technologies that support system adaptation. Sensor network applications were to be fielded and these applications evaluated. Our goal was to prove that this approach was: (*i*) feasible, (*ii*) more dependable than existing approaches, and (*iii*) efficient. Proposed deliverables were:

- Mobile code infrastructure
- Data fusion methodology
- Design support methodology
- Application implementations.
- Virtual Enterprise

This report documents RSN's results. It explains how we succeeded in accomplishing these goals.

3. MOBILE CODE SUPPORT FOR SENSOR NETWORKS

Sensor networks are built on the premise of a multitude of nodes working cooperatively. These nodes must be very inexpensive, and thus have severe resource constraints. Among these constraints are battery power and storage capacity. Mass storage is needed for maintaining both sensor data and systems software. Many reasonable applications require storing large amounts of data locally, leaving little room for software storage.

Traditionally embedded systems software has been very lean in order to save system memory and storage space. Once a system is in the field, it is traditionally impossible to modify its functionality without physically accessing the device. Only those tasks that are absolutely necessary are fielded and the system is immutable. In the context of sensor networks this means that in traditional systems:

- Each node has a fixed role.
- Its abilities are limited to a minimal subset of the desirable ones.
- Applications requiring target specific data and programs will be deployed with data and programs only for the target classes that are anticipated.

This severely limits system functionality.

We have implemented lightweight mobile code daemons to overcome these limitations. Nodes are deployed with the software suite they are most likely to require. Should they attempt to use a function that is not currently present on the node, the system automatically locates another node containing the desired program, downloads the code and executes it. If there is not enough storage space currently available on the node, a garbage collection routine can be run to delete programs that are not currently needed. In many ways this functions like distributed cache or virtual memory systems. A hard resource limitation has been replaced with almost unlimited virtual storage. The cost of this is the power that will be expended transferring programs between nodes.

In exchange for this cost, node software can now be reconfigured dynamically. Node roles can change as required. Rarely used functionality is accessible as needed. Nodes can

be deployed years in advance of their use and target classification routines can be updated before or during use. Enemy forces may acquire new armaments after system deployment. Friendly assets could be captured making tracking target classes that were previously innocuous essential.

Each node runs a mobile code daemon. The daemon is lightweight with a small memory profile. It can push or pull data files. It is capable of sending, receiving, and/or executing programs. Programs can be executables, shared objects, dynamic link libraries, or interpreted files. The daemons interact as peers. Some peers provide indexing information to locate programs as needed. The number of indexes can vary from one per network to one per node. This allows the system to emulate the peer-to-peer (P2P) functionality of applications like Napster and Gnutella.

One unique aspect of our approach is *distributed dynamic linking*. At run time, the system is capable of choosing between different implementations of a program. For example, during system development our network consisted of nodes running Linux on both SH4 processors and X86 processors. An SH4 node requesting the same program as an X86 node would receive a different executable.

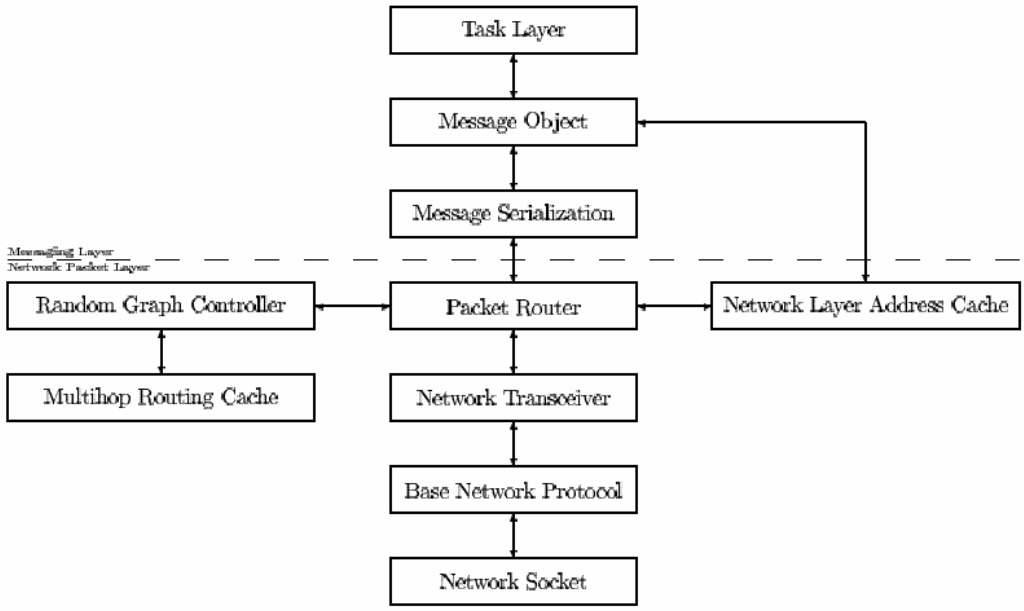

Figure 1. *Distributed Dynamic Linking*

The mobile code daemon is based upon the Remote Execution and Action Protocol (REAP). This protocol is responsible for message passing between nodes within our network. On top of this packet protocol we have developed a framework to allow objects to serialize themselves and travel across the network. At a higher layer of abstraction we have written messages to handle remote process creation and monitoring, simple file system operations, and resource index operations.

The daemon is written in C++. Versions were written for Windows NT, Windows CE , and Linux operating systems to conform with experimental plans for the SensIT WINS nodes. The daemon structure is broken down into several core modules: foundation classes,

the networking core, the random graph module, the messaging core, the packet router, the index server, the transaction manager, the resource manager, and the process manager. We will discuss each of these components in turn.

Before discussing the REAP daemon in detail, it is useful to discuss the underlying framework on which it is built. The framework abstracts many of the complexities of systems programming out of the core, into a set of libraries. Thus, we have written our own object-oriented threading and locking classes, whose current implementation calls into the threads library of the underlying operating system. We also rely heavily on a set of templated, multithreaded linked list, hash, and heap objects throughout the code. In addition, there are classes to handle singleton objects, the union-find problem, and object serialization. Lastly, there is also a polymorphic socket library that allows different networking architectures to emulate unicast stream sockets, regardless of the underlying network protocol or topology. These socket libraries are explained in the discussion of the networking core.

The daemon is capable of communicating over several networking technologies. The major ones are: TCP/IP, Diffusion Routing, and UNIX domain sockets. The Diffusion routing interface was written to allow interaction with other SensIT programs. The socket framework is designed so that new protocols are easily inserted into the daemon. To achieve this, an abstract base class 'Socket' includes all of the familiar calls to handle network I/O. Furthermore, all nodes are assigned a protocol-independent unique address. Opening a new socket involves looking up the network-layer address of a node in a local cache, and then opening the lower-level socket. When a cache miss occurs, a higher-level protocol is provided to find the network-layer address. The appropriate socket object is allocated based upon the network-layer address of the destination.

Diffusion provided some interesting challenges because it is not a stream-oriented unicast protocol. Rather, it provides a publish and subscribe interface, and is essentially a multicast datagram protocol. Thus, we had the choice of rewriting the REAP socket protocol as a datagram protocol, or building a reliable stream protocol on top of the Diffusion framework. It was deemed simpler to write a reliable stream protocol on top of Diffusion. In essence, we wrote a simplified userspace TCP stack. The current userspace stack employs the standard three-way handshake protocols for socket open and close, and it also employs a simple delayed-ACK algorithm. This system is implemented as an abstract child of the 'Socket' base class. Our Diffusion driver then provides an implementation of our userspace TCP module. The Diffusion driver performs a role equivalent to the IP layer processing code in most kernels. It receives datagrams from the Diffusion daemon through callback functions, parses the headers to make sure the datagram has reached the correct destination, and then either discards the contents, or passes it up to the TCP layer. These steps were deemed necessary because Diffusion is a multicast protocol, and thus we could not rule out the possibility of datagrams reaching our socket object that were not actually destined for it.

Early on in the project, it became clear that persistent connections between the various nodes was essential. A single file transfer of a shared object could result in thousands of packets traversing the network, and session setup time was simply too long over TCP and

Diffusion. To counteract this problem we implemented a system in which sockets are kept open whenever possible. The first implementation of this system opened directly to a destination, and did not support multi-hop routing very well. Under this implementation, socket timeout counters were employed to close underutilized sockets. This method has inherent scalability problems, and we decided a better solution was required.

This better solution involves a multi-hop packet routing network built on top of a random graph of sensor nodes. Each node in the system has four graph parameters specified: minimum degree, maximum degree, cliquishness, and clique radius. The cliquishness parameter defines the probability of a new edge being formed to a node within the clique radius. The minimum degree and maximum degree parameters control how many neighboring nodes can exist at any point in time. The clique parameters allow us to control the size and connectedness of cliques within the graph. Cliques become more important when we investigate the index system.

To add a new edge, a random number is generated to decide whether or not to add a clique edge. Then a random node from the node cache is chosen based upon two filter criteria: the chosen node must have a minimum path length of two to this node, and its minimum path length must be less than or equal to the clique radius for a clique edge, or greater than the clique radius for a non-clique edge.

The messaging system implements the core of the REAP protocol. At its lowest levels, this consists of a packet protocol, on top of which serialized objects are built. The 'Packet' class is nothing more than a variable-sized opaque data carrier that is capable of sending itself between nodes. The header defines enough information to route packets, specify the upper-level protocol, and to handle multi-packet transmissions where the number of packets is known a priori. The options field consists of a 4-bit options vector, and a 4-bit header extension size parameter. The TTL field is used in the new multi-hop protocol to eventually destroy any packet routing loops that might form.

Higher level messaging functionality is handled by a set of classes that do object serialization, and by a base message class. The serialization class in REAP provides a fast method of changing common data types into network byte-ordered, opaque data. The key advantage to this serialization system is that it only handles common data types, and thus has much lower overhead than technologies such as XDR and ASN.1.

The base messaging class provides a simple interface to control destination address, source transaction information, possible system state dependencies for message delivery, and control over sending the message. In addition, it defines abstract serialization and reordering functions that are implemented by all message types.

The serialization class sits beneath the base message class and does the physical work of serializing data, packetizing the serialized buffer, and then injecting those packets into the router.

On the receiving end, packets are received by an object serialization class and inserted into the proper offset in the receive buffer. A union-find structure keeps track of packet sequence numbers, and once it detects that all packets have been received, the message is delivered to a message queue in the destination task structure.

Another interesting feature of the messaging system is the function called 'run'. This function takes a task structure as an argument, and is generally intended to perform some action on the destination of the message.

The daemon packet router has several key responsibilities. The primary one is to use its internal routing tables to move packets from source to destination. The other primary function of the router is to coordinate the dissemination of multi-hop routing data.

The current method of determining multi-hop paths is through broadcast query messages. We gradually increase the broadcast TTL until a route is found, or a TTL upper limit is reached, at which point the node is assumed down. This methodology helps to reduce flooding, while making optimal paths likely to be found. A simple optimization allows a node to answer a multi-hop query if it has an answer in its routing table. Although this system is essentially a heuristic, it tends to work well because failed intermediate nodes are easily bypassed when their neighbors find that they cannot reach the next hop. Of course, this can lead to much longer paths through the graph, but support is integrated to warn of intermediate node failures, and multi-hop cache expire times help to reduce this problem by forcing refreshes occasionally. The multi-hop refreshes are carried out in unicast fashion, and a broadcast refresh is only used if a significant hop count increase is detected.

The actual routing of packets involves looking at the two destination fields in the packet header. First, a check is performed to determine whether the destination node identifier is equivalent to the current node's identifier, or the local loopback address, or one of several addresses that are defined for special purposes, such as broadcast to all members of a clique. The next check is to determine whether the destination process identifier is equivalent to that of the current process. If it is not, then the packet will need to be forwarded across a unix domain socket. If both of these tests pass, then the packet must be delivered to the appropriate task. Because packets do not contain sufficient routing data to deliver them to a specific task, we must recreate the high level message object in the router to determine the message's final destination.

Every task in a REAP process registers itself with the router during initialization. Once a task is registered, it can receive messages bound for any active ticket. Several special tickets are defined for every task that handle task status messages, and task-wide requests. Other tickets are ephemeral, and are allocated as needed.

An important component of the REAP daemon is the index system. This system implements a distributed database of resources available on the network. Each record in this database describes an object of one of the following types: index server, file, executable, library, pipe, memory map, host, or a task. Every record in the database has a canonical name, and resource locator associated with it. Both of these values are stored as human-readable strings. Besides this, metadata to allow for both data and metadata replication are present. The goal is to have a distributed cluster of index servers that transparently replicate each other's index records, and to have a resource control system that transparently replicates the actual data as well. At this point, the replication technology is only partially implemented.

The index system consists of the following modules: client, server, database, and the associated messaging protocol. The client is responsible for building a query message, sending the message, and either waiting for a response, or returning a response handle to the client in the case of an asynchronous call. The server consists of a pool of threads that poll for incoming messages on the server task structure. When a thread receives a message, it runs the query embedded in the message against the local database, and then sends the results back to the client in a query result message.

The index infrastructure is mainly built upon two message types: a query message, and a result message. The query message consists of an operand tree, some query option flags, and possibly a list of index records. Once the query message reaches the server, it is received by a server thread, and the 'run' function is called. This function performs a query against the index database object, and sends back a result message to the source node. Once these actions are complete, the 'run' function returns, and then the index server deallocates the query object. The index server itself is nothing more than a pool of threads that accept a certain type of message, and then allow the messages to perform their actions. In this sense, the REAP messaging system implements the mobile agent paradigm.

The other major feature of the index system is a system to select code based upon destination system architecture and operating system. To handle this, system architecture and operating system are considered polymorphic class hierarchies. Every index record contains an enumeration defining its membership in each hierarchy. When a system requests object code or binary data, we must ensure that it is compatible with the destination system. Thus, every index query can filter based upon architecture, if desired. When a query indicates that architecture and/or operating system are a concern, then C++ ‘dynamic_cast' calls are made to ensure compatibility. Because we are using the C++ dynamic casting technology, supported architectures and operating systems are determined at compile time. It would not be a technically difficult modification to use human-readable strings, and runtime-defined polymorphic hierarchies. However, we chose the compile-time approach because it is faster, and the architectures and operating systems in our lab are relatively constant.

The resource management framework is tightly coupled with the index system. When a client program wants to access a resource, a query to the index system is made. The results returned can then be passed into the resource management object. The resource manager then attempts to open one of the resource from the result set. If possible, one resource from each canonical name in the result set will be opened. Thus, the resource manager is capable of overcoming node failures by looking for other copies of the same resource. The current implementation attempts to open one instance of every canonical name in parallel, and continues this iterative process as timeouts occur. Eventually, an instance of every canonical name will be opened, or the resource manager will run out of instances of a resource in the index result set.

The resource control system is built on top of a client-server framework. This framework was chosen because the types of resources we want to support are generally not concurrent objects. Thus, the resource management system consists of two REAP message types: a resource operation message, and a resource response message. Then, there are two

types of resource objects: a client object, and a server object. For any given resource, there will exist exactly one server object, and one client object per task with an open handle to the resource. When a given client wants to perform an operation on the resource, it will send a resource operation message to the server object's transaction address. The server will then call the 'run' method of the message, and through a set of polymorphic calls described below, it will perform I/O operations on the server object. A response message will then be sent to the originating node.

The client and server resource objects are based upon an abstract interface that defines several common methods that can be used on UNIX file descriptors. The major base operations are: open, close, read lock, write lock, unlock, read, write, and stat. In all cases, blocking and non-blocking versions of these functions are provided, and the blocking functions are simply built on top of the non-blocking code.

The last major component of the REAP framework is process creation and management. This portion of the architecture consist almost entirely of message types. The primary message type is a process creation message. This message contains an index record pointing to the binary to execute. It also contains the argument and environment vectors to include, as well. A second message is process creation response message. This message simply contains the transaction address of the newly created process. Finally, task monitoring messages may be used to monitor the progress of a task using the publish/subscribe model discussed in the section on transaction management.

The REAP mobile code daemon permits us to experiment with many different mobile code paradigms over a fault-tolerant multi-platform framework. Because it provides a simple cross-platform, distributed interprocess communication framework, it is very useful for developing systems of collaborating distributed processes. This approach is capable of mimicking all the major mobile code paradigms, as shown in [Orr 2002]. Furthermore, its polymorphic code selection system permits us to use the optimal algorithm on a given system without significant user interaction. Finally, the distributed resource management system allows us to reduce bandwidth and permit concurrent use of resources without breaking normal concurrency rules.

4. MOBILE CODE API

The following interface was implemented for use within the SensIT community:

The node responds to the following API calls (semantics explained later):

status = **exec(***program-class*, *input-data-vector*, *output-data-vector, resource-list ,optional-command-line*)

status = **exec_noblock**(*program-class*, *input-data-vector*, *output-data-vector, resource-list, optional-command-line*)

status = **pipe(***program-class*, *input-data-vector*, *output-data-vector, resource-list, optional-command-line*)

status = **kill_pipe**(*program-class, machine*)

status = **lock**(*program-class, machine*)

status = **unlock**(*program-class*, *machine*)

status = **load**(*program-class*, *machine)*
status = **register_program**(*class, URL, optional-command-line)*
status = **register_machine**(*machine, port*)
status = **list_machines**(*machine-info-vector*)
status = **list_classes**(*class-info-vector*)
status = **list_available_classes**(*machine, class-info-vector)*
Parameters:

status - An integer value indicating call success or failure. ARL_MCN_SUCCESS indicates success. Other values indicate errors. A list of error returns will be provided with the final documentation.

program-class - In the initial delivery this will be a string name that uniquely identifies a program.

input-data-vector - A null terminated array of pointers to strings giving a list of URL's indicating files to be used as input.

output-data-vector - A null terminated array of pointers to strings giving a list of URL's indicating files to be used as output.

resource-list - A null terminated array of pointers to strings of resource identifiers. In the initial implementation, this will be a node name (ex. strange.arl.psu.edu).

machine - A string containing a node name (ex. strange.arl.psu.edu).

optional-command-line - A string defining the command line format for an executable or parameters for a DLL.

port - IP port number of the socket used by the ARL mobile code software to listen for mobile code requests.

machine-info-vector - A null terminated array of pointers to a data structure consisting of pointers to two fields: node_name, and port number.

class-info-vector - A null terminated array of pointers to a data structure consisting of pointers to three fields: class_name, URL, and default command line (possibly null).

Following is the call semantics for the mobile code package:

exec - Executes in four phases: (1) Uploads all data in the vector *input-data-vector* and the program *program-class.* (If files are not currently located on the node.) (2) Executes the program. (3) Writes output to the URLs in *output-data-vector*. (4) performs garbage collection. This call blocks until execution is complete. Returns the completion status of the program. The optional command line argument can be used to override the defaults given in *register_program.*

exec_no_block - Same as exec, but does not block.. Returns a completion status indicating system acceptance (or non-acceptance) the call. The optional command line argument can be used to override the defaults given in *register_program*

pipe - Executes in five phases: (1) Uploads the program *program-class* (If the file is not currently located on the node.) *input-data-vector* identifies files on the node. (2) Executes the program. (3) Writes output to the URLs in *output-data-vector*. (4) Waits for modifications to files in *input-data-vector* and then goes back to step (2). (5) Performs garbage collection on receipt of the out of band signal from kill_pipe(). This call does not block. Returns a completion status indicating system acceptance (or non-acceptance) of the call. The optional command line argument can be used to override the defaults given in *register_program*

kill-pipe - Sends an out of band message to *machine* indicating the pipeline should be terminated.

lock - Download a program to a node. Make this program unavailable for garbage collection.

unlock - Make a program previously unavailable for garbage collection, available for garbage collection.

load - If a local node attempts to execute a program not present locally, it can use this call to trigger a network interrupt. First nodes in the neighborhood will be signaled. If they have a copy of the program they will transfer the copy to the requesting node. Otherwise, the network interrupt will propagate through the multi-hop network up to the repository. If the program is found on a node on any of the hops, propagation of the request will stop and the program will be transmitted to the requestor. If the program is found on the repository, the program will be transmitted to the requestor. If the program is not found in the repository, an error condition is signaled.

register_program - Creates a link in the repository between the URL, the default command line and the unique class name. The default command line is either a string giving the command line arguments used when executing the class (including variables for input and output files), or a list of default parameters used when constructing a call to a function in a DLL. In the initial delivery, class name must be unique for each program. In later releases polymorphism and subclasses will be supported.

register-machine - Identifies a machine for use by the Mobile Code software.

list_machines - Returns a pointer to a machine-info-vector. One entry is given for every machine registered with the mobile code repository. This can be used to initialize a system list of nodes the machine can accept network connections from.

list_classes - Returns a pointer to a class info-vector. One entry is given for each class registered with the mobile code repository.

list_available_classes - Returns a pointer to a class info-vector. One entry is given for each class available for use on the node indicated by the *machine* parameter. If *machine* is null, the local node is used as a default.

5. CLASSIFIER SWAPPING

As a proof-of-concept demonstration of the mobile code daemon, we tested a system for swapping classification algorithms on the fly. This allows the sensor network to adapt to changes in its environment.

Target classification is a process in which sensor time-series data is used to assign target detections to one of a set of known classes. A vehicle driving by a sensor node could be a tank, a truck, a dragon wagon, a TEL, etc. Many classification techniques exist. In this test we used classifiers developed by Hairong Qi of the University of Tennessee at Knoxville, Akhbar Sayeed of the University of Wisconsin , and David Friedlander of the Penn State Applied Research Laboratory.

To choose between classifiers, confusion matrices were used. A confusion matrix is a matrix where the rows are the actual target classes and the columns are the predicted target classes. Each element of the matrix $e_{i,j}$ expresses the probability that the classifier returns codebook value j when target type i was actually present. For example, the matrix:

$$\begin{bmatrix} 0.90 & 0.10 \\ 0.25 & 0.75 \end{bmatrix}$$

could express the uncertainty in a classifier with two classes: tracked vehicle (class 1) and wheeled vehicle (class 2). In this case, the system correctly classifies class 1 90% of the time and class 2 75% of the time. In our case the code book values and target classes used were:

CODEBOOK	**Target Name**	**Target Descriptor**
0	Unknown	
10	sif_Buzzer	
11	sif_Motorcycle	Motorcycle
12	sif_TruckGas	Pickup Truck – Gas Engine
13	sif_TruckDiesel	Pickup Truck – Diesel Engine
14	sif_BuzzerRed	Red team
15	sif_BuzzerBlue	Blue team

Another part of the field test tested new classification techniques. Insufficient data was available before the tests to derive reliable confusion matrices for the three approaches used. Our classifier swapping tests used matrices fabricated to best illustrate the software functionality.

To support classifier swapping each node keeps a vector containing the set of target classes most recently detected by the sensor node.[1] The diagonal of the confusion matrix expresses the likelihood that a classifier is correct. The vector of target classes seen recently is multiplied against the diagonal of the confusion matrices for the three participating classifiers. This provides a measure of how well a given classifier should work given the current target mix.

[1] We implicitly assume that the types of targets the node is likely to see in the near future resembles what it has seen lately (locality in time and space).

All three participants used a unified classifier Application Programming Interface (API). This integrated the three different classifiers into the tracking process we developed for SensIT via a single call to the mobile code daemon. After each target was processed, the system determined which classifier was likely to work best with the current target mix. The daemon pre-fetches that classifier to be sure that it is present on the sensor node and uses it to classify the next target detected. This illustrates the distributed dynamic linking concept. A single classification call can trigger any of a number of implementations. The system automatically chooses the most appropriate one at run-time.

Implementation of this approach requires passing data to and from the classification program or library routine. The mobile code daemon can replace the routine used at will. We circumvent functionality traditionally given to the linker. We have used two different approaches to this problem: manufacturing call stacks and marshalling data to disk.

6. DISTRIBUTED TARGET TRACKING

The Collaborative Tracking Network (ColTraNe) is a fully distributed target tracking system. Sensoria Corporation constructed the sensor nodes used. Individual nodes use SH4 processors running Linux and are battery powered. Wireless communication for ColTraNe uses time division multiplexing. Data routing is done via the diffusion routing approach, which supports communications based on data attributes instead of node network addresses. Communications can be directed to geographic locations or regions.

Each node has three sensor inputs: acoustic, seismic, and passive infrared (PIR). Acoustic and seismic sensors are omni-directional and return time-series data. The PIR sensor is a two-pixel imager. It detects motion and is directional. Software provided by BAE Systems in Austin, Texas handles target detection. The software detects and returns Closest Point of Approach (CPA) events. CPA is a robust, easily detected statistic. A CPA event occurs when the signal intensity received by a sensor starts to decrease. Using CPA events from all sensor types makes combining information from different sensing modes easy. Combining sensory modes makes the system less affected by many types of environmental noise.

We summarize the ColTraNe process In Figure 2.

1. Each node waits for CPA events to be triggered by one or more of its sensors. The node also continuously receives information about target tracks heading towards it.
2. When a CPA event occurs, relevant information (node position, CPA time, target class, signal intensity, etc.) is broadcast to nodes in the immediate vicinity.
3. The node with the most intense signal in its immediate neighborhood and current time slice is chosen as the local *clump head*. The clump head calculates the geometric centroid of the contributing nodes' positions, weighted by signal strength. This estimates the target position. Linear regression is used to determine target heading, and velocity.
4. The clump head attempts to fit the information from step 3 to the track information received in step 1. We currently use a Euclidean metric for this comparison.

5. If the smallest such track fit is too large, or no incoming track information is found, a new track record is generated with the data from step 3. Otherwise, the current information from step 3 is combined with the information from the track record with the closest track fit to create an updated track record.
6. The record from step 5 is transmitted to the user community.
7. A region is defined containing the likely trajectories of the target; the track record from step 5 is transmitted to all nodes within that region.

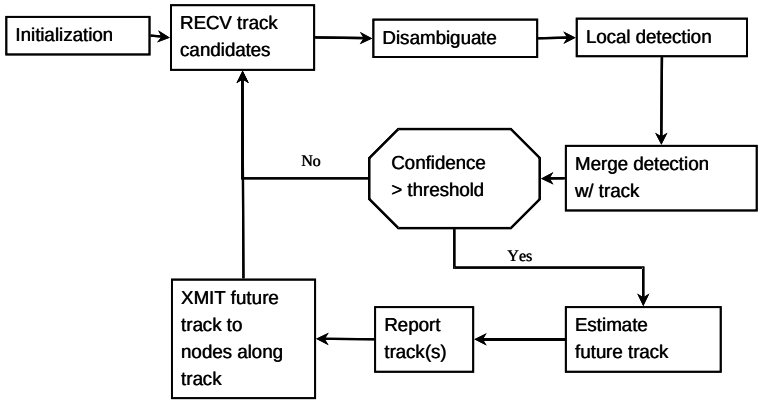

Figure 2. *Flowchart of the processing performed at any given node to allow distributed target tracking by ColTraNe*

Distributing logic throughout the network had unexpected advantages in our field test at Twenty Nine Palms in November 2001. During the test, hardware and environmental conditions caused 55% of the CPA events to be false positives. The tracks initiated by erroneous CPA events were determined by step 3 to have target heading and velocity of 0.0, thereby preventing their propagation to the rest of the nodes in step 7. Thus, ColTraNe automatically filtered this clutter from the data presented to the user. Problems with the Twenty Nine Palms implementation were discovered as well:

- Implementation schedule did not allow the EKF version of step 5 to be tested.
- Velocity estimation worked well, but position estimation relied on the position of the clump head.
- Tracks tended to branch, making the results difficult to decipher (see figure 3a).
- Tracking was limited to one target at a time.

Continued development has alleviated these problems. The Extended Kalman Filter has been integrated into the system. It improves the quality of both track and target position estimates as tracks progress.

An angle gate, which automatically excludes continuations of tracks when velocity estimates show targets are moving in radically different directions, has been inserted into the track matching metric. This reduces the tendency of tracks to branch, as shown in figure 3b.

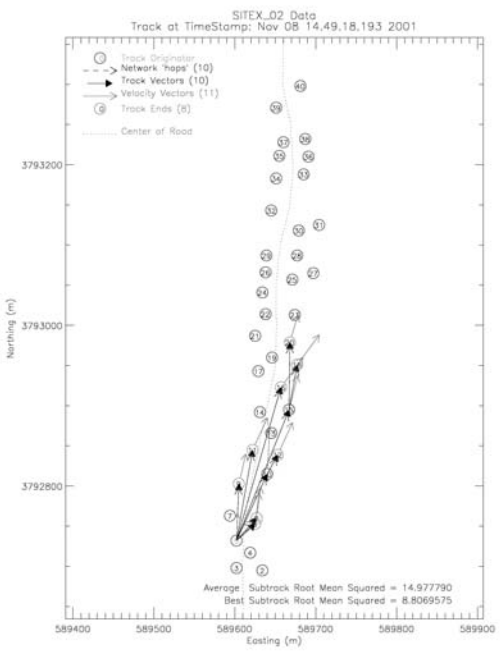

Figure 3a. No filtering

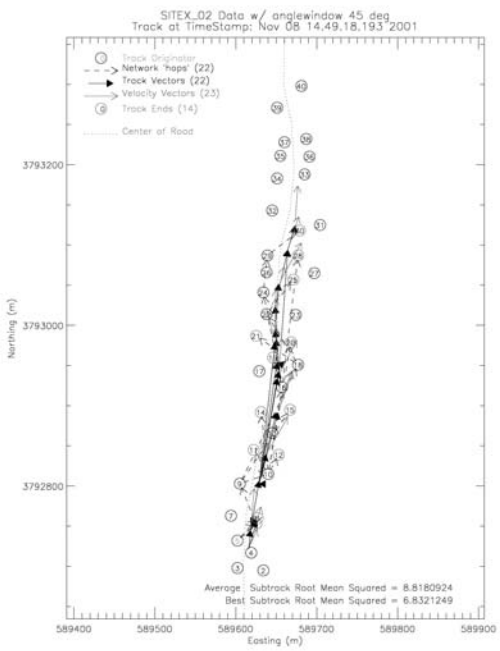

Figure 3b. 45-Degree Angle Filter

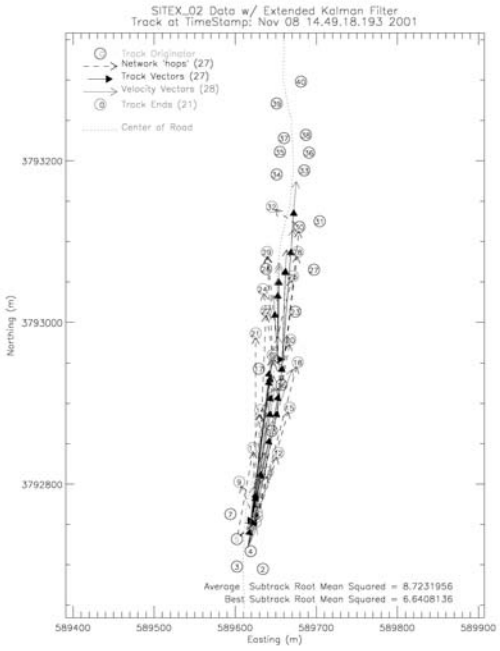

Figure 3c. Extended Kalman Filter

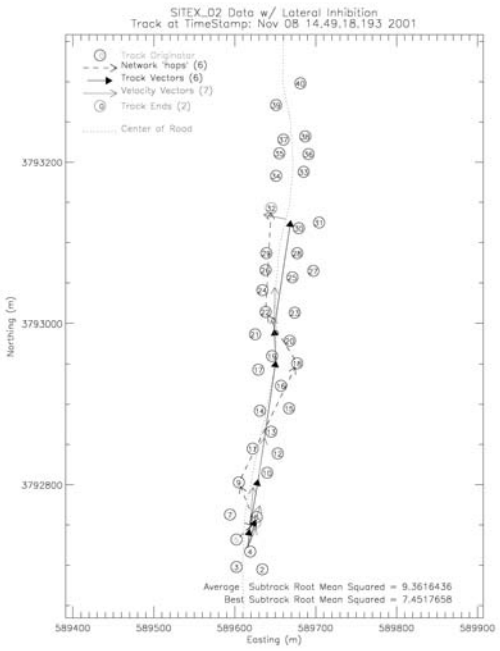

Figure 3d. Lateral Inhibition

Figure 3. *Both axes are UTM coordinates. Circles are sensor nodes. The faint curve through the nodes is the middle of the road. Dark arrows are the reported target tracks. Dotted arrows connect the* clump heads *that formed the tracks. Filtering not only reduced the systems tendency to branch, but also increased the track length.*

We constructed a technique for reducing the tendency of tracks to branch. We call this technique lateral inhibition. Before continuing a track, nodes whose current readings match a candidate track broadcast their intention to continue the track. They then wait for a period of time proportional to the log of their goodness of fit value. During this time, they can receive messages from other nodes that fit the candidate track better. If better fits are received, they drop their continuations. If no one else reports a better fit within the timeout period, the node continues the track. Figure 3d shows a target track with lateral inhibition.

The target position is now estimated as the geometric centroid of local target detections with signal intensity used as the weight. Our tests indicate this is more effective in improving the position estimate than the Extended Kalman Filter. The geometric centroid approach was used in the angle filter and lateral inhibition test runs shown in figure 3 and table 1.

Differences between the techniques can be seen in the tracks in figure 3. The tracks all use data from a field test with military vehicles at Twenty Nine Palms Marine Training Ground. Sensor nodes were placed along a road and at an intersection. In the test run depicted in figure 3, the vehicle traversed the sensor field along a road going from the bottom of the diagram to the top. The faint dotted line shows the position of the center of the road. Figure 3a diagram shows results from our original implementation. The other diagrams use our improved techniques and the same sensor data.

Figure 3a illustrates the deficiencies of our original approach. The tracking process works, but many track branches form and the results are difficult to interpret. Introducing a 45 degree angle gate (figure 3b) reduces track branching. It also helps the system correctly continue the track further than our original approach. Estimating the target position by using the geometric centroid greatly improves the accuracy of the track. This approach works well because it assumes that targets turn slowly and in this test the road section is nearly straight.

Using the Extended Kalman Filter (EKF) (figure 3c) also provides a more accurate and understandable set of tracks. Branching still occurs, but is limited to a region that is very close to the actual trajectory of the target. The EKF performs its own computation of the target position. Like the angle filter, the EKF imposes a linear model on the data, and hence works well with the data from the straight road.

The lateral inhibition results (figure 3d) have the least amount of track branching. This track is the most easily understood of all the methods shown. It is non-parametric and does not assume linearity in the data. As with the angle gate, the geometric centroid is a good estimate of the target position. We have also tested a combination of EKF and lateral inhibition. The results of that approach are worse than either the EKF or lateral inhibition approaches in isolation.

Our discussion of the track data is supported by the error data summarized in table 1. Each cell shows the area between the track formed by the approach and the actual target trajectory. The top portion of the table is data from all the tracks taken on November 8 2001. The bottom portion of the table is from the track shown in figure 3. In both portions, the top row is the average error for all the tracks formed by a target. The bottom row is the sum of all the errors for all the tracks formed by a target.

	RMS for tracks from Nov 08 2001				
	Live Data	Angle 45 Deg	EKF	Lateral Inhibition	EKF & LAT
Averaged	18.108328	9.533245	8.877021	9.361643	11.306236
Track Summed	81.456893	54.527057	52.775338	13.535534	26.738410
	RMS for track beginning at Nov_08_14.49.18.193_2001				
	Live Data	Angle 45 Deg	EKF	Lateral Inhibition	EKF & LAT
Averaged	14.977790	8.818092	8.723196	9.361643	8.979458
Track Summed	119.822320	123.453290	183.187110	18.723287	35.917832

Table 1. Root mean square error comparison for the data association techniques discussed. The top set of numbers is for all target tracks collected on the day of November 8. The bottom set of numbers is for one specific target run. In each set, the top row is the average error for all tracks made by the target during the run. The bottom row sums the error over the tracks. Since these tests were of a target following a road, the angle and EKF filters have an advantage. They assume a linear trajectory. Lateral inhibition still performs well, although it is non-parametric.

If one considers only the average track error, the EKF provides the best results. The original approach provides the worst results. The other three approaches considered are roughly equivalent.

Summing the error of all the tracks formed for a single target penalizes approaches where multiple track branches form. When this is done, lateral inhibition has the most promising results. The second best results are provided by the combination of lateral inhibition and EKF. The other approaches are roughly equivalent.

These results show that the inter-node coordination provided by lateral inhibition is a promising technique. Since it is non-parametric it makes very few assumptions about the target trajectory.

Geometric centroid is a robust position estimator. Robust local parameter estimation provides a reliable estimate of the target's position and heading. Lateral inhibition reduces the tendency of our original tracking implementation to produce confusing interpretations of the data inputs. The system finds the track continuation that is the best continuation of the last known target position. In combination, both methods track targets moving through a sensor field more clearly.

The distributed nature of this approach makes it very robust to node failure. It also makes multiple target-tracking problems easy to solve when targets are much more sparsely distributed than the sensors. Multiple target tracking becomes a disjoint set of single target tracking problems.

Multiple target tracking conflicts arise only when target trajectories cross each other or approach each other too closely. When tracks cross or approach each other closely, linear regression breaks down since CPA events from multiple targets will be used in the same computation. The results will tend to match neither track. The tracks will be continued once the targets no longer interfere with each other.

Classification algorithms will be useful for tracking closely spaced targets. If crossing targets are of different classes and class information is transmitted as part of the CPA event, then the linear regression could be done on events grouped by target class. In which case, target crossing becomes even less of a concern.

Table 2 compares the network traffic incurred by the approaches shown in figure 2 with the bandwidth required for a centralized approach using CPA data. CPA packets had 40 bytes, and the lateral inhibition packets had 56 bytes. Track data packets vary in size, since the EKF required three data points and a covariance matrix. The table shows that lateral inhibition requires the least network bandwidth due to reduced track divergence.

	CPA size	40			
	Inhibition size	56			
	Track packets	Track pack size	CPA packets	Inhib. packets	Total
EKF	852	296	59	0	254552
Lateral inhibition	217	56	59	130	21792
EKF & Lateral inhibit	204	296	59	114	69128
Centralized	0	0	240	0	9600

Table 2. Data transmission requirements for the different data association techniques. The total is the number of bytes sent over the network. The EKF requires covariance data and previous data points. Angle gating and lateral inhibition, require less data in the track record. Data is from the tracking period shown in figure 2.

Note from table 2, that in this case centralized tracking required less than half as many bytes as lateral inhibition. This data is somewhat misleading. The data shown is from a network of 40 nodes with an Internet gateway in the middle. As the number of nodes and the distance to the gateway increases, the number of packet transmissions will increase for the centralized case. For the other techniques, the number of packets transmitted will remain constant. Recall the occurrence of tracking filter false positives in the network, which was more than 50% of the CPA's during this test. Reasonably, under those conditions the centralized data volume would more than double over time and be comparable to the lateral inhibition volume. Note as well that centralized data association would involve as many as 24 to 30 CPA's for every detection event in our method. When association requires $O(n^2)$ comparisons [Bar-Shalom 1993] this becomes an issue.

To analyze the ability of ColTraNe to track multiple targets we performed the following experiment. Two simulated targets were sent through a simulated sensor node field comprised of 400 nodes arranged in a rectangular grid measuring 8 x 8 meters. Two different scenarios were used for this simulation.

1) X Path: Two targets enter the field at the upper and lower left corners, traverse the field crossing each other in the center of the field and exit at the opposite corners. See figure 4a.

2) Bowtie: Two targets enter the field at the upper and lower left corner, traverse the field along hyperbolic paths that nearly intersect in the center of the field, and then exit the field at the upper and lower right corners. See figure 4b.

Calculation of the tracking errors was accomplished by determining the area under the curve between a track plot and the target path to which it was related.

The Collaborative Tracking Network performed very well in the X pattern tests due to the linear nature of our track continuations. Tracks seldom jumped to the opposite target path and almost always tracked both targets separately. Bowtie tracking, however, turned out to be more complex. See figures 5 and 6.

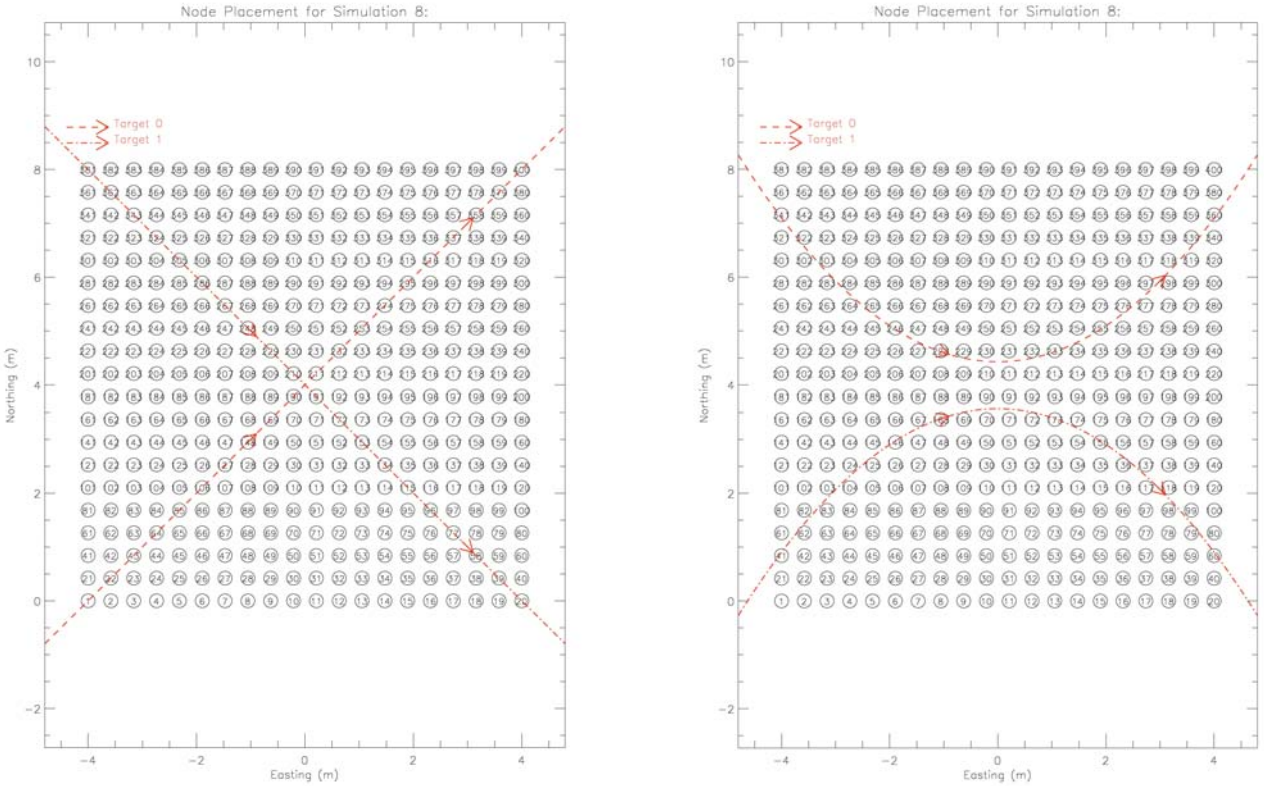

Figure 4a. X Path Simulation

Figure 4b. Bowtie Path Simulation

Figure 4. *Comparison of the two Multiple Target Tracking Simulation Scenarios. Circles are sensor nodes. The faint lines crossing the node field are the target paths.*

Bowtie target paths that approach each other too closely at point of nearest approach (Conjunction) tend to cause the tracks to cross-over to the opposite target path as if the targets' paths had crossed each other (figure 5). Again, this is due to the linear nature of the track continuations.

As Conjunction distance increases beyond a certain point (Critical Conjunction), the incidence of cross-over decreases dramatically (figure 6). Minimum Effective Conjunction is the smallest value of Conjunction where the incidence of cross-over begins to decrease to acceptable levels.

According to our analysis as shown in figure 7, if a Clump Range equal to the Node Separation is used, Critical Conjunction is equal to the Node Separation and Minimum Effective Conjunction is approximately 1.5 times Node Separation. If Clump Range is equal to 2 times Node Separation, Critical Conjunction is equal to 2.5 times Node Separation and Minimum Effective Conjunction is approximately 3 times Node Separation or 1.5 times Clump Range.

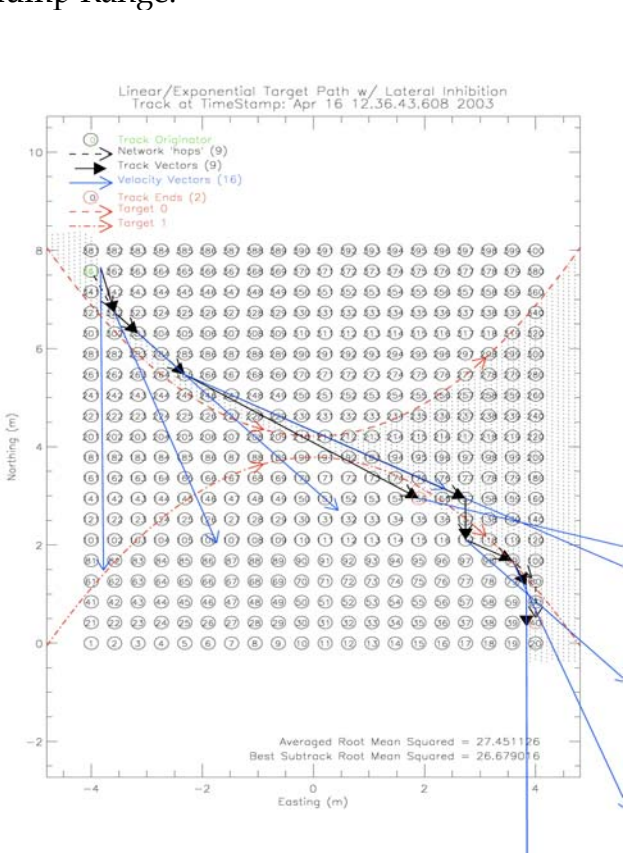

Figure 5a. Track for Target 1

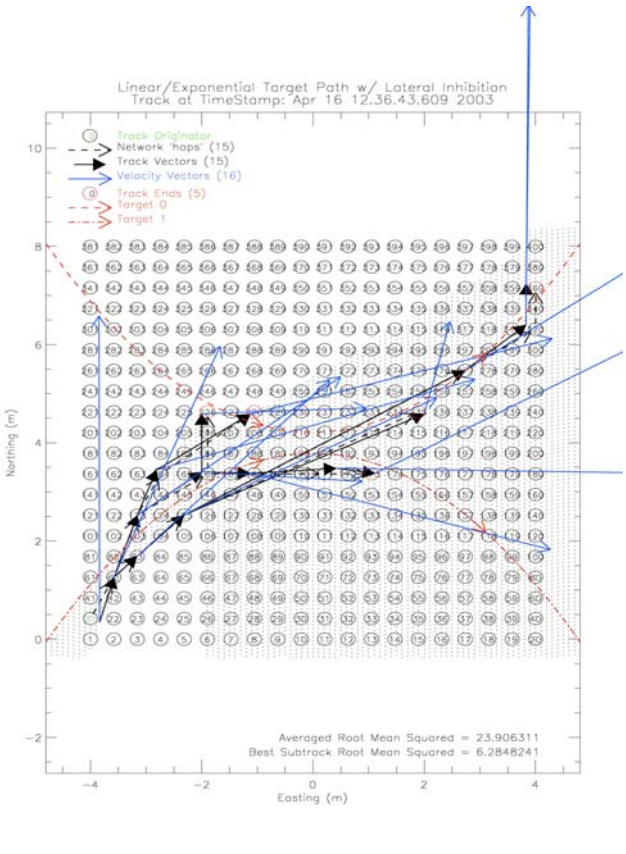

Figure 5b. Track for Target 2

Figure 5. *Bowtie Tracks for Conjunction equal to Node Separation Distance.* Dark arrows are the reported target tracks. Lighter arrows are the calculated velocity vectors. Shaded areas are the area between the curves used to determine track error. Other notations as for Figure 3.

The significant result of this analysis seems to be that the Minimum Effective Conjunction is equal to 1.5 times Clump Range. This means that ColTraNe should be able to independently track multiple targets provided they are separated by at least 1.5 times the Clump Range. This appears to be related to fundamental sampling limitations based on Nyquist Sampling Theory [Jacobson 2003].

Our work indicates to us that performing target tracking in a distributed manner greatly simplifies the multi-target tracking problem. If sensor nodes are dense enough and targets are sparse enough, the multi-target tracking is a disjoint set of single target tracking problems. Centralized approaches will also become untenable as target density increases.

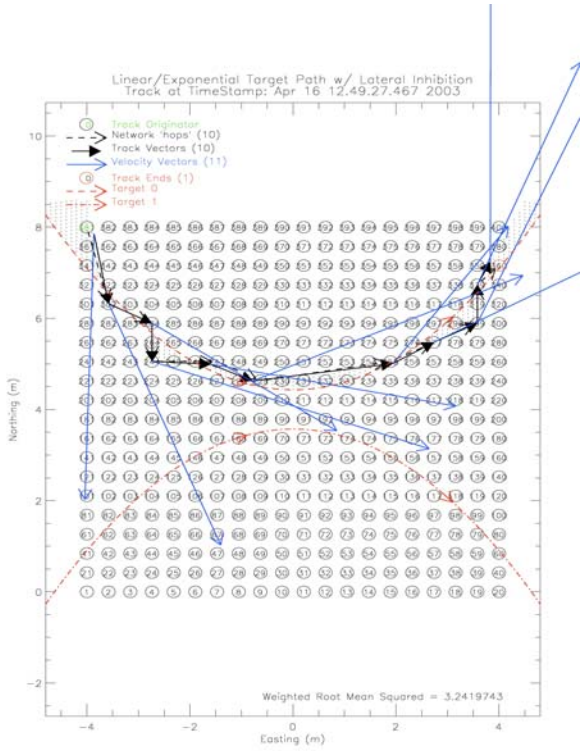

Figure 6a. Track for Target 1

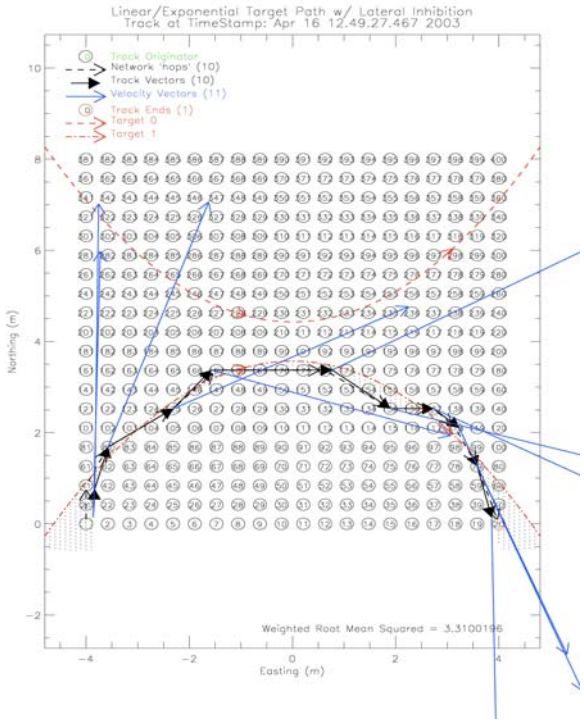

Figure 6b. Track for Target 2

Figure 6. **Bowtie Tracks for Conjunction equal to 2 times Node Separation Distance.** Notations as for Figure 5

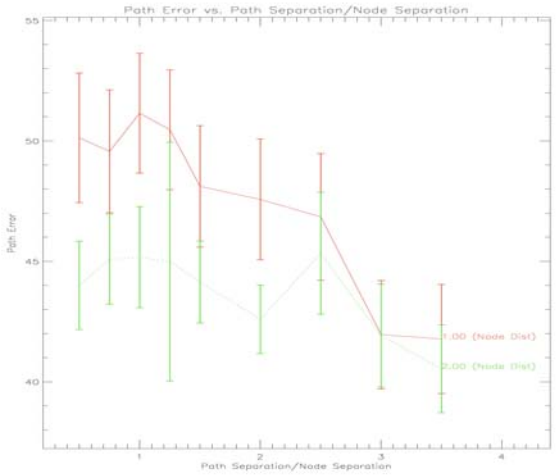

Figure 7. *Finding Critical Conjunction Experimentally.* ***The darker upper line displays the results when the Clump Range is equal to the Node Separation distance. The lighter lower line displays the results when Clump Range is equal to 2 times the Node Separation Distance.***

7. INTEGRATION OF MULTIPLE SENSING MODALITIES

The PI of this project received Defense University Research Instrumentation Program (DURIP) funding to create an augmented reality smart space, where multiple sensor inputs are combined in a networked environment. The networked environment is made up of multiple heterogeneous processors, including embedded processors. Communications uses both wired and wireless connections. Figure 8 provides a floorplan of the laboratory.

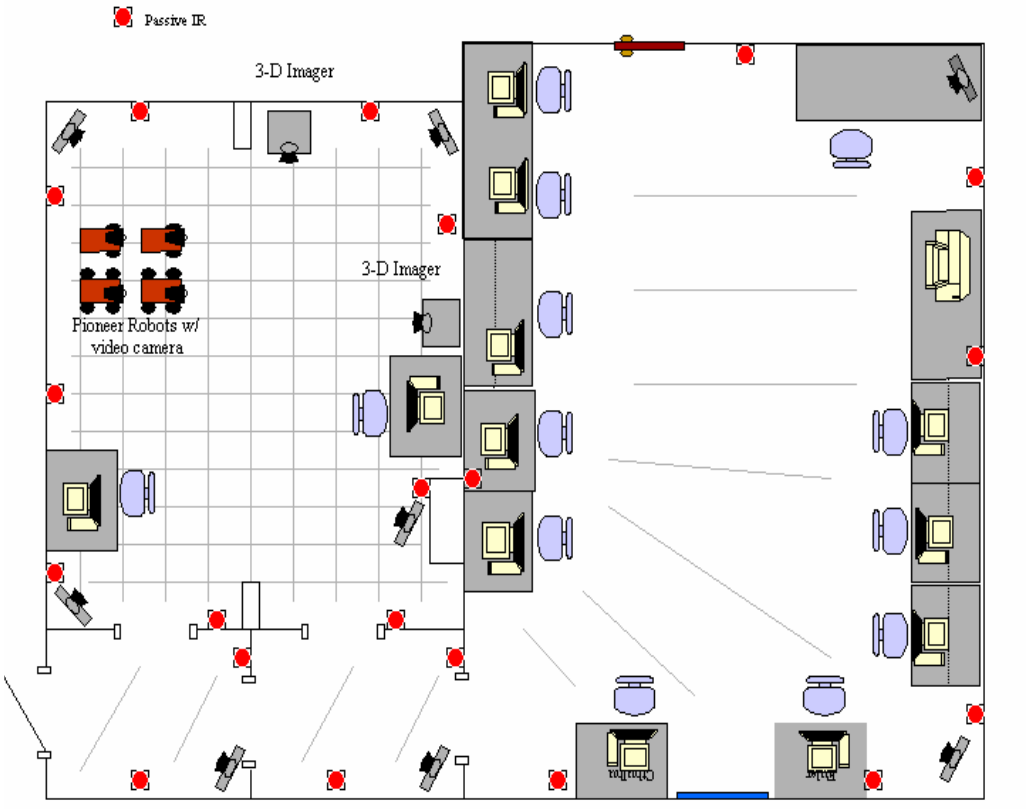

Figure 8. *Layout of sensor network laboratory.*

As part of the extension to RSN, we were tasked with extending our sensor network approach to sensing modalities that were not included in SensIT work to this point. Figure 9 shows examples of multiple target tracking using video inputs. Much of the logic is similar to the CPA based tracking we have discussed earlier. What is different is primarily the lower level signal processing primitives. In the examples given, we track people. The process is:

- Three different probability distributions (red, green, blue) are retained for each pixel in the image. Each probability distribution expresses the intensity variance of that color band within recent history.
- When pixel values in the RGB bands in the current image are outside the range of values that would be expected based on the current variance, the pixel is marked as being part

of the foreground. This allows the system to adapt to regular changes in the environment.

- Morphological processing removes groups of foreground pixels that are not large enough to be of interest. The black (background) and white (foreground) portions of figure 9 provide samples of this process.
- Bounding rectangles are computed for blobs of foreground pixels whose size could represent a human being. The images on the right in figure nine show bounding boxes around human targets.
- A track ID is associated with each bounding box.
- Tracking using the track ID essentially uses the same distributed logic discussed earlier.

Figure 9. *Examples of video tracking of pedestrians in the laboratory.*

Figure 10 shows an example of how we extended video tracking to integrate more sensing modalities. A camera cueing system was developed. The floor of the laboratory contains pressure sensitive plates. When a change in pressure is detected in the region a camera is pointed at, the camera is activated. This is an alternative method for tracking pedestrians. Note that our approach integrates infrared and normal video inputs. The laboratory also has an omnicamera video input that has a 360 degree field of view. Combining the perspectives from two cameras would allow triangulation to compute the absolute position of a target.

Figure 12 shows the results from a tracking server that was implemented to integrate multiple sensing modalities. The grid shown represents the laboratory environment as seen from above. The yellow rectangle shows a pressure sensitive plate that has detected a significant change in pressure. The red circle in the middle of the yellow rectangle is the current target position. Gray and red rectangles positioned on the perimeter of the field are passive infrared motion detectors, like those used at 29 Palms. Red ones have detected motion. Gray ones have not detected motion.

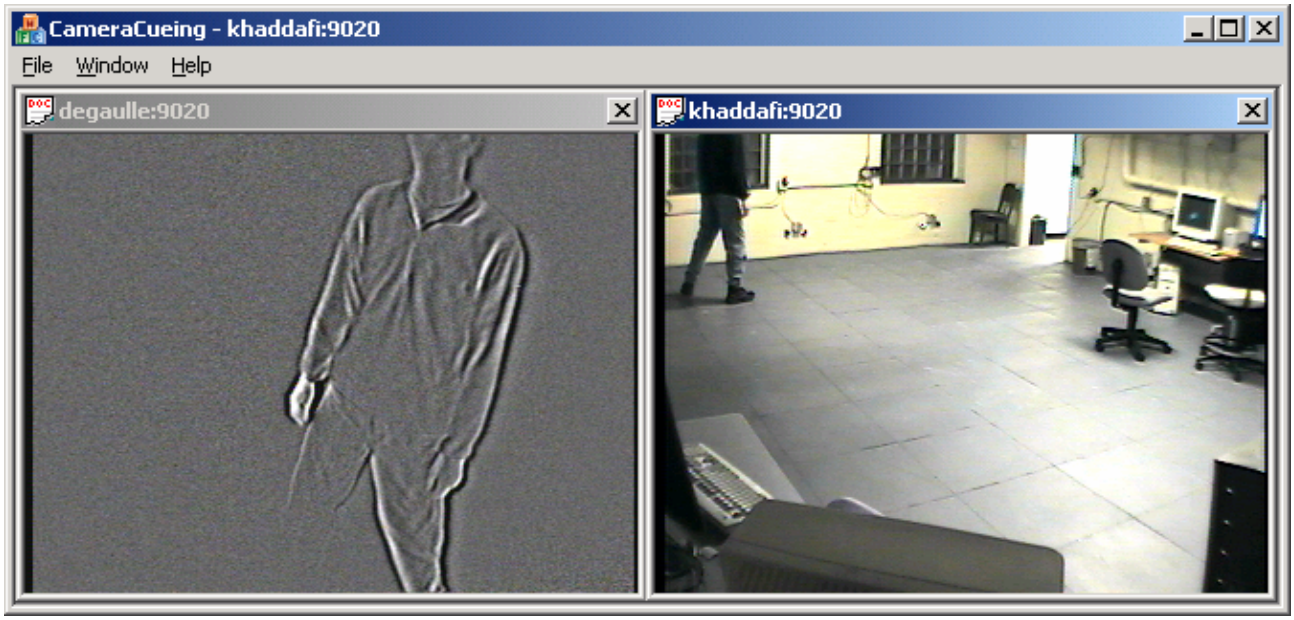

Figure 10. **As a pedestrian moves from one region of the lab to another, cameras are activated to monitor its progress.** Cameras include omnicamera, normal video, low-light IR (right), and true IR (left).

Figure 11. *360 degree panorama of the laboratory from a ceiling mounted fish eye lens camera.*

The room also has two ladar devices located on the walls at the top and bottom of the image. Their positions are marked by large blue rectangles. They send out a laser plane. This provides range information across a plane. The purple and white lines in the image show ladar readings. Note that both readings intersect at the target position. The red dotted lines indicate readings from sonar nodes. When the dotted lines end in a red rectangle, an obstruction has been detected. Figure 12 shows an example output from a tracking server that integrates all of these modalities. The process also uses windows, linux, and embedded processors using combinations of wired and wireless communications. In the laboratory all communications is IP based. In our previous work, we have shown our ability to translate this to other approaches, like diffusion.

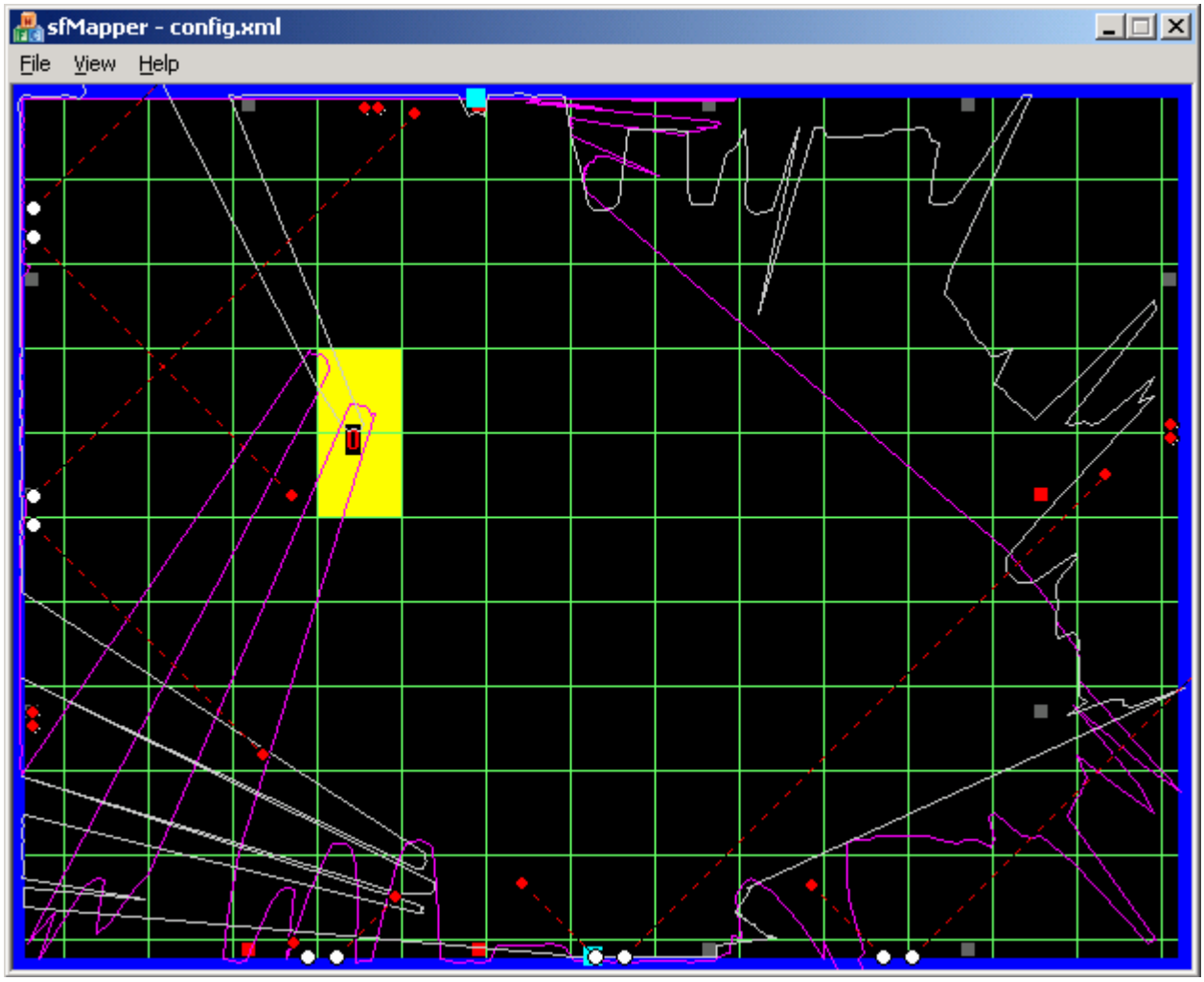

Figure 12. *Laboratory view of a target moving through the laboratory. Multiple sensing modalities are combined to detect and follow the target.*

8. DISTRIBUTED SYSTEMS DEVELOPMENT APPROACH

Sensor networks create a distributed information-processing environment. Sensor nodes in the networks we consider are self-contained battery-powered units with sensors, CPUs, storage, and wireless communications. Each node collects data samples from its environment; evaluates the data; and forwards evaluation results to other nodes.

For many reasons, it is advisable for a large number of nodes to cooperate in the network. Having multiple nodes decreases system vulnerability to single points of failure. Statistical data analysis is more reliable when a large number of data samples are available. Wireless communications are mainly short range due to power constraints and multi-path fading. Using a large number of nodes provides multiple paths for data flow, which makes the network more robust and increases data throughput.

Our approach uses cellular automata models to evaluate distributed application designs. These approaches distribute computation throughout the network. We use Cellular Automata (CA) tools to evaluate and contrast network embedded designs. Our approach combines recent approaches to traffic and urban planning.

CA traffic models look at granular media (automobiles) flowing through constrained pipes (roads). Simulations using CA tools mimic these models. They qualitatively reproduce many empirically observed consequences of complex interactions, such as the wave-like propagation of traffic jams. These CA models are known collectively as "particle-hopping" models. We use a particle-hopping model to study data packet propagation in sensor networks. Data packets form a granular medium that moves through bandwidth-limited communications channels.

Urban planning models expand CA's to include mobile agents that inhabit the CA. The "Free Agents in a Cellular Space" (FACS) simultaneously influence and are influenced by the CA they inhabit [Portugali 2000]. This extension makes it possible to express distributed coordination problems in a straightforward manner.

Our approach allows us to simultaneously measure and evaluate:

- Ability of the algorithms to perform their tasks.
- Resource requirements of different algorithms.
- Global implications of local behaviors.
- The ability of the system to organize itself.

Sensors detect entities in their vicinity and exchange information about the entities. The network as a whole continually provides its user community with up-to-date estimates of attributes describing the entities crossing the field.

Our approach is characterized by:

- Integrating multiple sub-tasks into a single framework.
- Network self-organization built on a publish-subscribe network paradigm.
- Entity detection, classification, and parameter estimation performed locally with locally available information.
- Data association and ambiguity resolution done locally.
- Prediction of future trajectory of the path done locally.

- Local publication of current estimates for consumption by the user community.

All approaches have a self-referential component. Each node makes local decisions based on locally available information. Other nodes are recruited for participation based on their position relative to the local node. This removes the need for central coordination and greatly increases the scalability of the approach. It also increases the system's robustness, since the system is able to adapt naturally to limited node failures. Under some conditions sub-optimal decisions may be made due to the myopic nature of this approach.

Evolving distributed systems have been modeled using Cellular Automata (CA) [Wolfram 1994]. A CA is a synchronously interacting set of abstract machines (network nodes). A CA is defined by:

- d the dimension of the automata
- r the radius of an element of the automata
- δ the transition rule of the automata
- s the set of states of an element of the automata

An element's (node's) behavior is a function of its internal state and those of neighboring nodes as defined by δ. The simplest instance of a CA has a dimension of 1, a radius of 1, a binary set of states, and all elements are uniform. In this case for each individual cell there are a total of 2^3 possible configurations of a node's neighborhood at any time step—if the cell itself is considered part of its own neighborhood. Each configuration is expressed as an integer v:

$$v = \sum_{i=-1}^{1} 2^{j_i+1} \tag{1}$$

where i is the relative position of the cell in the neighborhood (left=-1, current position =0, right=1), and j_i is the binary value of the state of cell i. Each transition rule can therefore be expressed as a single integer r known as its Wolfram number [Wolfram 1994]:

$$r = \sum_{v=1}^{8} j_v * 2^v \tag{2}$$

where j_v is the binary state value for the cell at the next time step if the current configuration is v. This is the most widely studied type of CA. It is a very simple many-to-one mapping for each individual cell.

Four complexity classes have been defined for these models. In the uniform class, all cells eventually evolve to the same state. In the periodic class, cells evolve to a periodic fixed structure. The chaotic class evolves to a fractal-like structure. The final class shows an interesting ability to self-organize into regions of local stability. This ability of CA models to capture emergent self-organization in distributed systems is crucial to our study.

We use more complex models than the ones given by equations (1) and (2). CA models have been used successfully to study traffic systems and mimic qualitative aspects of many problems found in vehicular traffic flow. For example they can illustrate how traffic jams propagate through road systems. By modifying system constraints, it is possible to create systems where traffic jams propagate either opposed to or along the direction of traffic flow. This has allowed physicists to empirically study how highway system designs influence the flow of traffic.

Many of the CA models are called "particle-hopping" models. The most widespread particle-hopping CA model is the Nagel-Schreckenberg model. This is a variation of the one-dimensional CA model expressed by equations (1) and (2). This approach mainly considers stretches of highway as a one-dimensional CA. It typically models one lane of a highway. The highway is divided into sections, which are typically uniform. Each section of the highway is a cell. The sizes of the cells are such that the state of a cell is defined by the presence or lack of an automobile in the cell. All automobiles move in the same direction.

With each time step, every cell's state is probabilistically defined based on the states of its neighbors. Nagel-Schreckenberg's CA is based on mimicking the motion of an automobile. Only one automobile can be in a cell at a time, since two automobiles simultaneously occupying the same space causes a collision. If an automobile occupies a cell, the probability of the automobile moving to the next cell in the direction of travel is determined by the speed of the automobile. The speed of the automobile depends on the amount of free space in front of the automobile, which is defined by the number of vacant cells in front of the automobile. In the absence of other automobiles (particles), an automobile moves at maximum speed along the highway by hopping from cell to cell. As more automobiles enter the highway congestion occurs. The distance between particles decreases and consequently speed decreases.

We adapt this approach to modeling sensor networks. Instead of particles representing automobiles moving along a highway, they represent packets in a multi-hop network moving from node to node. Each cell represents a network node rather than a segment of a highway lane. Since we are considering a two-dimensional surface covered with sensor nodes, we need a two-dimensional CA. The cells are laid out in a regular matrix. A node's neighborhood consists of the eight nodes adjoining it to the North, South, East, West, Northwest, Northeast, Southwest and Southeast. For this paper we assume that nodes are fixed geographically—e.g. non-mobile. A packet can move from a node to any of its neighbors. The number of packets in the cell's node defines the cell's state. Each node has a finite queue length. A packet's speed does not depend on empty cells in its vicinity. It depends on the node's queue length. Cell state is no longer a binary variable; it is an integer value between 0 and 10 (chosen arbitrarily as the maximum value).

As with Nagel-Schreckenberg, particle (packet) movement from one cell to another is probabilistic. This mirrors the reality that wireless data transmission is not 100% reliable. Atmospheric and environmental affects, such as sunspots, weather, and jamming can cause packets to be garbled during transmission. For our initial tests, we have chosen the information sink to be at the center of the bottom edge of the sensor field. Routing is done by sending packets along the shortest viable path from the sensor source to the information sink, which can be determined using local information. Paths are not viable when nodes in the path can no longer receive packets. This may happen when a node's battery is exhausted, or its queue is full.

This adaptation of particle-hopping models is suitable for modeling the information flow in the network; however, it does not adequately express sensing scenarios where a target traverses the sensor field. To express scenarios we have included "Free Agents in a Cellular Space" (FACS) concepts from [Portugali 2000]. Portugali uses ideas from

Synergetics and a CA including agents to study the evolution of ethnic distributions in Israeli urban neighborhoods. In the FACS model, agents are free to move from cell to cell in the CA. The presence of an agent modifies the behavior of the cell, and the state of a cell affects the behavior of an agent.

In our experiments, entities traversing the sensor field are free agents. They are free to follow their own trajectory through the field. Detection of an entity by a sensor node (cell) triggers one of the entity tracking algorithms. This causes track information to be transmitted to other nodes and to the information sink. Network traffic is a non-linear phenomenon. Our model integrates network traffic analysis into the tracking algorithm simulations. This includes propagating traffic jams as a function of sensor network design.

The existence and rate of growth of traffic jams around the sink is a function of the rate of information detection and the probability of successful data transmission. Consider false detections in the sensor grid, where p is the false alarm probability. For small p, no traffic jams form. If p increases beyond a threshold p_c, traffic jams form around the sink. The value of p_c appears to be unique to each network. In our model it appears to be unique to each set of CA transition rules. This result is consistent with queuing theory analysis where maximum queue length tends to infinity when the volume of requests for service is greater than the system's capacity to process requests.

Traffic flux through the sink is proportional to the number of tracks being monitored, the probability of false detection and the probability of a successful transmission. Assuming a perfect network and non-ambiguous tracks, this relationship is linear e.g.

$$\varphi_{\sin k} = kT$$

Where T is the number of tracks and φ is the flux. Track ambiguities and networking imperfections cause deviations from this linear structure. The exact nature of the distortion depends directly on the CA transition rule and the type of track uncertainty.

The primary purpose of this part of the project was to create a methodology for designing distributed systems that interact with the real world. CA models allow us to construct models based on the interaction of autonomous nodes. These models include system faults and network traffic. We posit that this type of analysis is important for the design of robust distributed systems, like autonomous sensor networks.

Simple cellular automata can be classified into four equivalence classes. Two dimensional traffic modeling CA are more difficult to classify. The cellular behavior may be periodic, stable and chaotic in different regions of the CA in question. Exact classification may be impossible or inappropriate.

We have shown that for certain probabilities of false positives stable traffic jams will form around the sink location; while for other values unstable traffic jams form. These are traffic jams that continue to form, disappear and reform. This oscillatory behavior is typical of periodic behavior of cellular automata. It is possible to have a stable traffic jam with an unstable boundary.

In the target-tracking context, we have established strong and weak points of the algorithms used. Pheromones appear to be robust, but transmit more data than the other algorithms. They can also be fooled by false positives.

The Bayesian network is effective for reducing the transmission of false positives but has difficulty in maintaining track continuation. Most likely, further work is required to tune the probabilities used.

EKF tracking may not be appropriate for this level of analysis, since it is designed to overcome Gaussian noise. At this level of fidelity that type of noise is less important. The CA model is discrete and the EKF is meant for use with continuous data.

Hybrid approaches may be possible and desirable. One possible avenue to consider is using the Bayesian logic to restrict the propagation of pheromones or to analyze the strength of the pheromone concentration present.

Our tracking algorithm development will continue by porting these algorithms to an NS [11] based sensor network simulation of higher fidelity. A live network test is planned for Fall 2003 at a government facility. It would be worthwhile to study the ability of the CA approach to predict behaviors in both the higher fidelity NS simulation, and live networks.

9. SENSOR NETWORK VIRTUAL ENTERPRISE

Our technology allows virtual enterprises to be constructed on the fly. Nodes' roles are decided dynamically. This significantly increases system robustness by allowing the system to adapt to the failure of individual nodes. The nodes that remain exchange readings and find answers. In this section, we discuss the dependability implications of this approach. We use the application we fielded at 29 Palms as an example.

Since the heading and velocity estimation approach uses triangulation, at least three sensor readings are needed to get an answer. In the following, we assume all nodes have an equal probability of failure q. In a non-adaptive system when the "cluster head" fails, the system fails. The cluster has a probability of failure q no matter how many nodes are in the cluster. In the adaptive case, the system fails only when the number of nodes functioning is three or less.

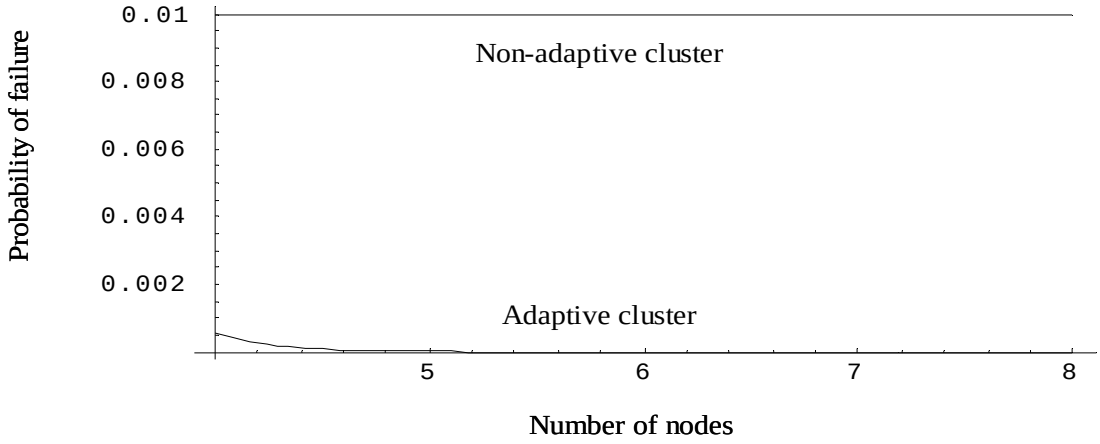

Figure 13. **The top line shows probability of failure for a non-adaptive cluster.** The bottom line shows probability of failure for an adaptive cluster. The probability of failure for a single node q is 0.01. The number of nodes in the cluster is varied from 4 to 8.

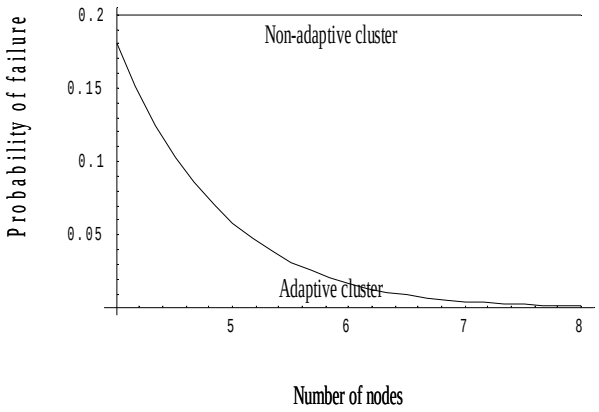

Figure 14. **The top line shows probability of failure for a non-adaptive cluster.** The bottom line shows probability of failure for an adaptive cluster. The probability of failure for a single node *q* is 0.2. The number of nodes in the cluster is varied from 4 to 8.

Figures 13-15 illustrate the difference in dependability between adaptive and non-adaptive tasking. These figures assume an exponential distribution of independent failure events, which is standard in dependability literature. The probability of failure is constant across time. We assume that all participating nodes have the same probability of failure. This does not account for errors due to loss of power.

In figures 13 and 14 the top line is the probability of failure for a non-adaptive cluster. Since one node is the designated cluster head, when it fails the cluster fails. By definition, this probability of failure is constant. The lower line is the probability of failure of an adaptive cluster as a function of the number of nodes. This is the probability that less than three nodes will be available at any point in time. All individual nodes have the same failure probability, which is the value shown by the top line. The probability of failure of the adaptive cluster drops off exponentially with the number of nodes. Figure 15 shows this same probability of failure as a function of both the number of nodes and the individual node's probability of failure.

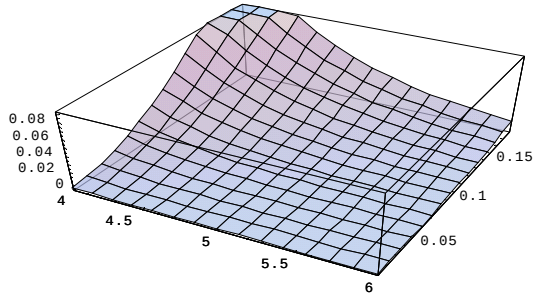

Figure 15. *The surface shows probability of failure (z axis) for an adaptive cluster as the probability of failure for a single node* q *varies from 0.01 to 0.2 (side axis), and the number of nodes in the cluster varies from 4 to 6 (front axis).*

10. DISCUSSION

This report provides technical details concerning the work performed on the DARPA IXO SensIT Reactive Sensor Networks project. The project explored a new approach to sensor network design and implementation.

A design methodology was created that was inherently self-referential. Each node worked relative to its own position and based on its own observations. In doing so, we also consider potential environmental factors and failure modes. This provides a simple viewpoint for algorithm design. It also allows us to develop simulations to explore the space of feasible algorithms. The very abstract simulations allow brittle approaches to be identified early in the design process. Detailed design work can then concentrate on the approaches with a stronger likelihood of success.

Nodes are also allowed to change their roles dynamically. To support this, we implemented mobile code daemons. The daemons support autonomous software reconfiguration. Analytical results were given that showed how this creates a more robust, dependable infrastructure. Software releases were provided to allow this approach to be used by the wider SensIT community.

To illustrate the feasibility of this approach, we fielded a distributed target tracking system. The multiple target tracking problem was analyzed in detail. We found that the quality of the tracks we produced was limited by data sampling. The sensor density determines the amount of data that is available for track inference. This limitation is basically is the same thing as the Nyquist critical frequency.

We also created a framework for configuring node software so that target classification methods can be changed in response to the observed mix of targets. The network as a whole is thus able to reconfigure its software and adapt to its environment.

As part of our contract extension, the target tracking approach was generalized. We showed how many different sensing modalities could be integrated into the overall framework we proposed.

11. CUMULATIVE LIST OF PUBLICATIONS SUPPORTED BY THIS GRANT

The following publications attributed to RSN have been published, are under review, or are in press:

"Data Registration," by R. R. Brooks and Lynne Grewe, published as chapter 6 of *CRC Handbook of Sensor Fusion*. David Hall, editor.

"SenseIT - Sensor Information Technology," by R. R. Brooks, published as chapter 25 of *CRC Handbook of Sensor Fusion*. David Hall, editor.

R. Brooks, et al. "Reactive Sensor Networks: Mobile Code Support for Autonomous Sensor Networks," *Distributed Autonomous Robotic Systems DARS 2000,* pp. 471-472, Springer Verlag, Tokyo.

R. R. Brooks. "Stigmergy an intelligence metric for emergent distributed behaviors." *NIST Workshop on Performance Metrics for Intelligent Systems.* August 14-16. Gaithersburg, MD.

R. R. Brooks, Lynne Grewe, and S. S. Iyengar, "Recognition in the Wavelet Domain: a survey" *Journal of Electronic Imaging*. Published August, 2001.

J. Moore, T. Keiser, R. R. Brooks, S. Phoha, D. Friedlander, J. Koch, A. Reggio, and N. Jacobson, "Tracking Targets with Self-Organizing Distributed Ground Sensors," *2003 IEEE Aerospace Conference*, Invited Paper, March 2003.

R. R. Brooks, P. Ramanathan, and A. Sayeed, "Distributed Target Tracking and Classification in Sensor Networks," *Proceedings of the IEEE*, Invited Paper, In Press.

R. Brooks, Friedlander, E. Grele, C. Griffin, N. Jacobson, T. Kaiser, J. Koch, S. Phoha, J. Moore, and T. Reggio, "Distributed Tracking and Classification of Land Vehicles by Acoustic Sensor Networks," *Journal of Underwater Acoustics*, Classified Journal, Invited Paper, Accepted for publication, April 2003.

R. Brooks, C. Griffin, and D. S. Friedlander, "Self-Organized distributed sensor network entity tracking," *International Journal of High Performance Computer Applications*, special issue on Sensor Networks, vol. 16, no. 3, pp. 207-220, Fall 2002

R. Brooks and C. Griffin, "Traffic model evaluation of ad hoc target tracking algorithms," *International Journal of High Performance Computer Applications*, special issue on Sensor Networks, Vol. 16, no. 3, pp. 221-234, Fall 2002.

S. S. Iyengar and R. R. Brooks, *Frontiers in Distributed Sensor Networks*, CRC Press, Boca Raton, FLA, in press, planned publication Fall 2003. Chapters 2.6 Target tracking and 6.4 Mobile code support of the book are attributed to RSN.

R. Brooks, D. Friedlander, J. Koch and S. Phoha, "Tracking Multiple Targets with Self-Organizing Distributed Ground Sensors," Submitted for review, Special Issue *Journal of Parallel Data Processing*, Brooks and Iyengar, ed.

12. LIST OF PERSONNEL ASSOCIATED

Dr. Richard R. Brooks - PI - Head, Distributed Intelligent Systems Dept. Applied Research Laboratory of The Pennsylvania State University.

Dr. Shashi Phoha - Co-PI - Head, Information Science and Technology Division, Assistant Director of Applied Research Laboratory, and Professor of Electrical and Computer Engineering. Applied Research Laboratory of The Pennsylvania State University.

Dr. Eric Keller – Research Associate - Distributed Intelligent Systems Dept. Applied Research Laboratory of The Pennsylvania State University.

Dr. Wojcieck Klimkiewicz – Research Associate - Distributed Intelligent Systems Dept. Applied Research Laboratory of The Pennsylvania State University.

Mr. Jason Douglas – Research Engineer. Applied Research Laboratory of The Pennsylvania State University.

Mr. Eric Grele - Research Engineer. Applied Research Laboratory of The Pennsylvania State University.

Mr. Christopher Griffin – Research Engineer. Applied Research Laboratory of The Pennsylvania State University. Currently pursuing a masters degree in mathematics

Mr. John Koch - Research Engineer. Applied Research Laboratory of The Pennsylvania State University.

Mr. Anthony Reggio - Research Engineer. Applied Research Laboratory of The Pennsylvania State University.

Ms. Jamila Moore - Graduate Assistant. Applied Research Laboratory of The Pennsylvania State University. Currently pursuing a masters degree in electrical engineering.

Mr. Matthew Pirretti - Graduate Assistant. Applied Research Laboratory of The Pennsylvania State University. Currently pursuing a Ph. D. degree in computer science and engineering.

Mr. Brian Ecker – Undergraduate. The Pennsylvania State University. Computer Science and Engineering.

Mr. Thomas Keiser – Undergraduate. The Pennsylvania State University. Computer Science and Engineering.

Mr. Brian Kovac – Undergraduate. The Pennsylvania State University. Computer Science and Engineering.

13. PRESENTATIONS

The following papers were presented:

- J. Moore, T. Keiser, R. R. Brooks, S. Phoha, D. Friedlander, J. Koch, A. Reggio, and N. Jacobson, "Tracking Targets with Self-Organizing Distributed Ground Sensors," *2003 IEEE Aerospace Conference*, Invited Paper, March 2003.
- R. Brooks, et al. "Reactive Sensor Networks: Mobile Code Support for Autonomous Sensor Networks," *Distributed Autonomous Robotic Systems DARS 2000*. Oak Ridge, TN. October 2000.
- R. R. Brooks. "Stigmergy an intelligence metric for emergent distributed behaviors." *NIST Workshop on Performance Metrics for Intelligent Systems.* August 2000. Gaithersburg, MD.

14. INVENTIONS

Three new inventions have been developed. We have submitted invention disclosures and performed patent searches for the discoveries. They have been declared to the university and evaluated by ARL internally for possible patent protection. They are:

REAP - *Remote Execution and Action Protocol* - This protocol is a central part of our mobile code infrastructure. It allows multiple requests consisting of several actions to be active simultaneously. It minimizes the number of ACK and NAK packets required to ensure remote execution of tasks. The protocol is complete. The invention disclosure has been

submitted. ARL performs a quarterly internal review to determine whether or not to pursue patent protection of inventions.

Nonlinear Differential Equations for Modeling Network Traffic - The process is documented. The invention disclosure has been written. ARL performs an internal review quarterly to determine whether or not to pursue patent protection. This process is very important for modeling ad hoc networks where transient effects are more important than asymptotic performance.

Distributed Dynamic Linking and Polymorphism - Extension of .DLL concepts to a network environment, and ability of the network to automatically resolve hardware dependencies.

15. PROGRAM FINANCIAL SUMMARY

Total Project

	Actual	Budget
Salary	$328,747.08	$ 337,427.00
Fringe Benefits	$70,431.29	$ 83,175.00
Travel	$31,932.86	$ 13,804.00
Equipment	$8,037.51	$ 7,188.00
Tuition	$24,699.58	$ 21,903.00
Overhead	$189,924.01	$ 190,150.00
Col. Total	$653,772.33	$ 653,647.00

16. REFERENCES

[Bar-Shalom 1993] Y. Bar-Shalom and X-R. Li, *Estimation and Tracking: Principles, Techniques, and Software*. Artech House, Boston, 1993.

[Cebrowski 1998] A. K. Cebrowski and J. J. Garsta, "Network-Centric Warfare: Its Origin and Future," *Proceedings of the Naval Institute,* Jan 1998. http://www.usni.org/Proceedings/Articles98/PROcebrowski.htm.

[Czerwinski 1998] T. Czerwinski, *Coping with the Bounds: Speculations on Nonlinearity in Military Affairs,* National Defense University, Washington, DC.

[Jacobson 2003] N. Jacobson, *Target Parameter Estimation in a Distributed Acoustic Network*, Honors Thesis, The Pennsylvania State University, Spring 2003

[Orr 2002] Nathan Orr, *A Message-Based Taxonomy of Mobile Code for Quantifying Network Communication*, M.S. Thesis, Penn State Computer Science and Engineering, Summer 2002.

[Portugali 2000] J. Portugali, *Self-Organization and the City*, Springer Series in Synergetics, Springer Verlag, Berlin, 2000.

[van Crefeld 1985] M. L. van Crefeld, *Command in War,* Harvard University Press, Cambridge, MA,1985.

[Wolfram 1994] S. Wolfram, *Cellular Automata and Complexity*. Addison-Wesley, Reading, MA, 1994.